Project Title: 8-bit Signed Serial-Parallel Multiplier on Artix-7 FPGA (Basys 3)

Course Title: Digital Design
Course Number: CSCE 2301
Semester: Spring 2025
Instructor: Mohamed Shalan

Student Names:
Rawan Khalid Muhammad
ID: 900232389
Doha Deia Nour El-Din
ID: 900232246
Noor Emam
ID: 900222081

Design Outlining

1.1. System Overview

The Serial-Parallel Multiplier (SPM) project implements a sequential digital circuit capable of multiplying two 8-bit signed binary numbers using a hybrid architecture: serial for the multiplier and parallel for the multiplicand. This approach helped in balancing between hardware simplicity and acceptable speed. The entire system is developed and deployed on the Basys 3 FPGA board using Verilog HDL. The design optimizes performance and area by integrating Carry-Save Adders (CSA) for efficient partial product accumulation and a dedicated Two's Complement (TCMP) unit for signed arithmetic correction. It uses toggle switches to input the two operands, push buttons for controlling operations, and 7-segment displays for presenting the signed decimal output. A finite state machine (FSM) is used to control the system's workflow including data loading, multiplication steps, and completion indication.

All modules are described in Verilog HDL and deployed on the Basys 3 FPGA.

Major Blocks:

1. **Multiplier Shift Register (8-bit):** Stores the multiplier input. This register shifts right on each clock cycle to provide one bit at a time for the multiplication process.

2. **Product Register (16-bit):** Accumulates the partial product results during the operation. It stores the final signed multiplication result.

3. **Input Interface:** X[7:0] for multiplier, Y[15:8] for multiplicand.

4. **CSA (Carry Save Adder):** Used for efficient partial product accumulation without immediate carry propagation. It accelerates addition of partial products during the multiplication.

5. **TCMP (Two's Complement Unit):** Handles sign extension and negation for signed multiplication. Converts negative binary numbers to two's complement format before processing and re-adjusts the sign afterward.

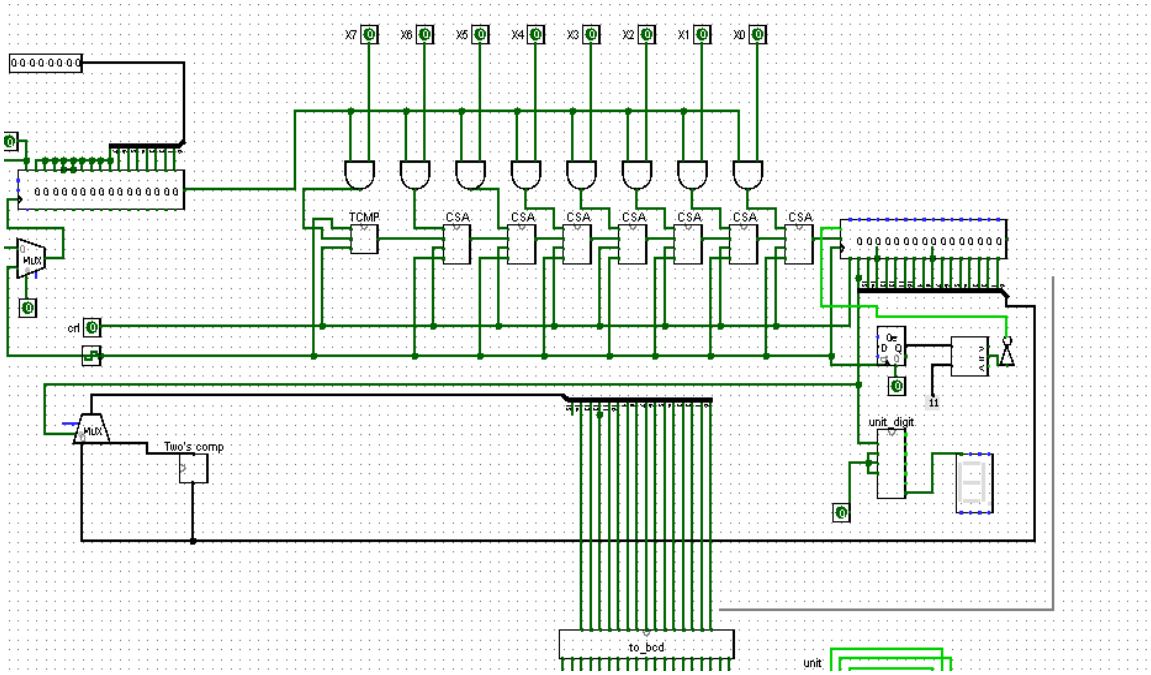
6. **Sign Logic:** Determines the sign bit of the result based on the original signs of the multiplier and multiplicand. Displays the sign on the leftmost 7-segment digit.

7. **to_bcd (Binary to BCD Converter):** Converts the 16-bit binary product to BCD format using Double Dabble algorithm or add_3 method for driving the display.

8. **Unit Digit Extractor**:
   Extracts individual decimal digits from the BCD result to be mapped onto each 7-segment display position.

9. **Two's complement:** If the sign bit for the result is 1 then we will do two's complement to get the positive decimal beside the sign.

10. **Output Interface:** 7-segment display for signed decimal product, LD0 for done flag.



1.2. Component Descriptions

1. Multiplier Register (Serial)

- A right-shifting register that stores the 8-bit multiplier.
- After each cycle, the register shifts right by one bit.

2. Multiplicand Register (Parallel)

- Holds the 8-bit signed multiplicand value loaded in parallel from SW [7:0].
- The value remains fixed during the 8-cycle multiplication process.

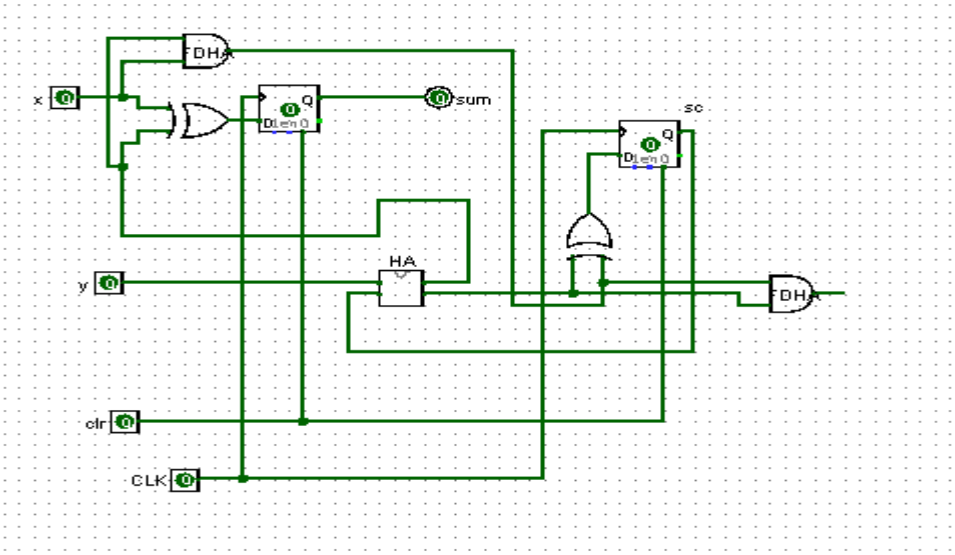
3. Product Register

- A 16-bit register that accumulates the partial results of multiplication.
- Starts at zero and updates conditionally on each clock cycle.
- Uses the MSB to check if the number negative to do the 2's complement

4. Carry-Save Adder (CSA)

- During each clock cycle, when the multiplier's LSB is 1, the multiplicand is added to the current accumulated value using the CSA technique.

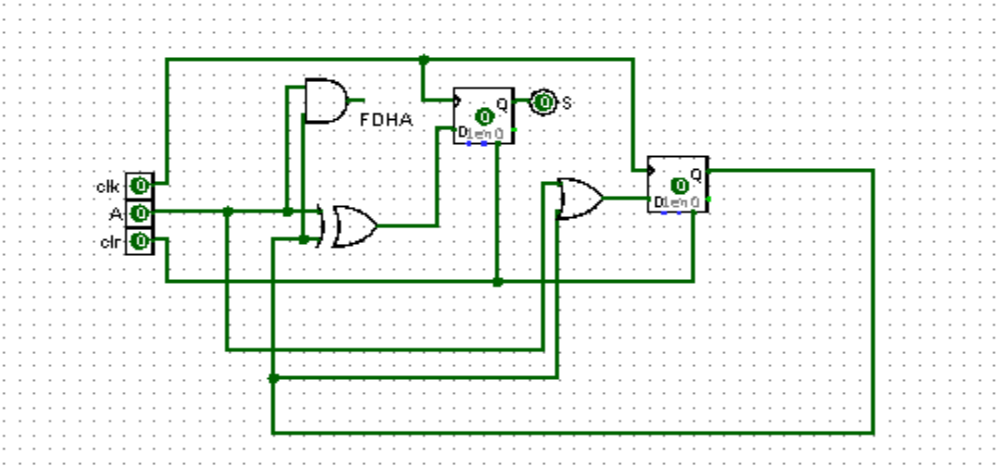
- The CSA structure avoids carry propagation delays by separating sum and carry values, which are resolved only at the end, making sequential additions faster than traditional ripple-carry adders.



5. Sign Detection and Correction using TCMP

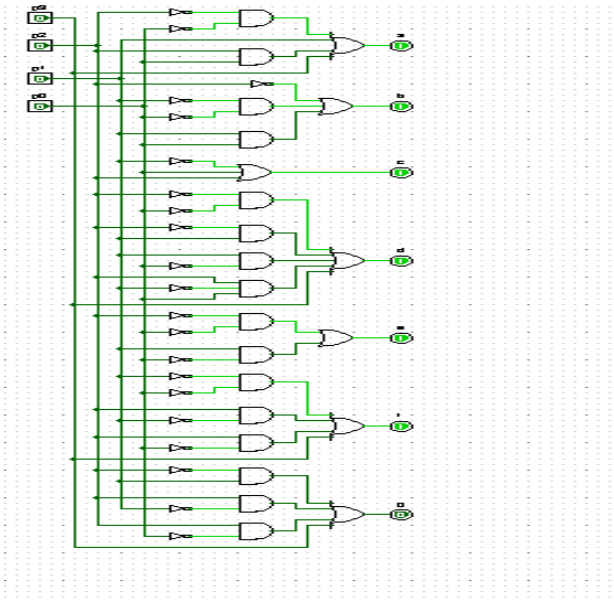
- The system handles signed inputs using Two's Complement (TCMP) logic. As both operands are initially interpreted as 2's complement numbers.
- Before starting the multiplication loop:
    1. The sign of the final result is computed using XOR of the two operand signs.
    2. If either operand is negative, it is converted to its magnitude using a TCMP module.

- After the CSA-based accumulation, if the product is expected to be negative, the final result is converted back into 2's complement form using the TCMP unit again.



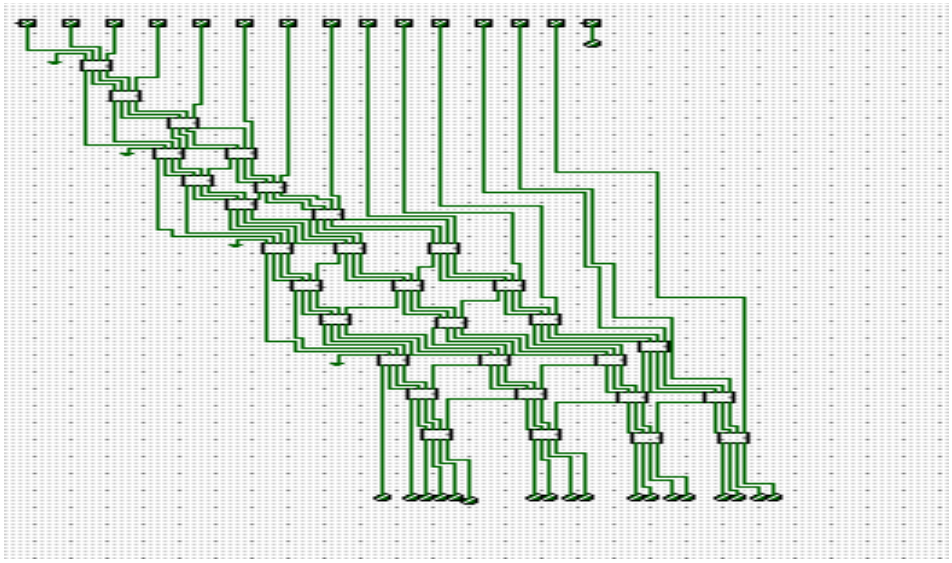
6. Unit Digit:

This logic circuit is used to take a 4-bit binary number (often from a counter or user input) and convert it into signals that light up a 7-segment display in the shape of the corresponding decimal digit (0–9). It only works for inputs from 0000 (0) to 1001 (9), which are valid BCD inputs.



7. Binary to BCD Conversion: add_3 and to_bcd Modules

1. add_3: Core logic unit used in the Double Dabble algorithm to correct digits during BCD conversion. Adds 3 to any 4-bit value ≥5 before shifting.
2. to_bcd: Implements the complete Double Dabble algorithm to convert the 16-bit binary product into 5-digit BCD, which can be displayed on 7-segment displays.
   Using Add 3, we build the to_bcd to add 3.



Display Formatting:

- The sign is output on the leftmost 7-segment display digit.

1.6 Input and Output Interface

Input Controls:

- SW[7:0]: Multiplier (8-bit signed).
- SW[15:8]: Multiplicand (8-bit signed).
- BTNC: Start multiplication.
- BTNL: Scroll product display left.
- BTNR: Scroll product display right.

2. Implementation Issues and Strong points

Implementing the Serial-Parallel Multiplier (SPM) on an FPGA using Verilog introduced several hardware and design-related challenges. The project involved designing the multiplier and integrating display modules and control logic to interact with user input via buttons. The following issues were encountered during the development process.

2.1 Sign Extension

Initially, sign extension was not applied to the operands, which led to incorrect results when multiplying negative numbers. This was because the higher-order bits were not properly extended to reflect the sign, causing the multiplier to interpret the values as unsigned and producing incorrect partial products. After identifying this issue during simulation and testing, the design was modified to correctly apply sign extension before performing the multiplication. Specifically, the sign bit (most significant bit) of the operand was replicated across the extended higher bits to preserve the correct signed value. In Verilog, this was done using the syntax {{8{y[7]}}, y}, which ensures that the operand is correctly sign-extended to match the internal processing width. With this fix, the SPM was able to produce accurate results for both positive and negative inputs.

2.2 Shifting Buttons

Another issue encountered during implementation was that the right and left buttons were not successfully scrolling the displayed result on the 7-segment displays. Initially, pressing the buttons produced no visible change, even though the shift logic was correctly implemented. After debugging, the problem was found to be due to the clock divider not being consistent across all modules. This led to timing mismatches where the shift signals were either missed or not registered correctly by the display module. To resolve the issue, a single, shared clock divider was created and distributed to all modules that required a slower clock. This ensured that the button detection, shift logic, and display updates were all synchronized to the same timing domain. Once this correction was made, the right and left scrolling functions worked reliably, confirming that the root cause was inconsistent clocking rather than logical errors in the shift mechanism.

2.3 Incorrect Output

Initially, the output of the multiplication is wrong. Discovering the problem was demanding. To discover a problem, we debug every part of the code. We made the state as output to see whether it is changing or

not. This debugging part was used in the module “FSM”. Another way we used to check that each module separately was working correctly and to detect which module has the issue was to make a testbench for each module separately, like the FSM and to_bcd modules. After checking all the testbenches, we realized that some modules are working correctly and others’ not. We realized that there is a problem in applying two’s complement and also in the FSM module, which is the sign extension, as previously mentioned. Correcting these issues made the output as expected, and the problems were solved.

2.4 Complex binary-to-BCD Converter

The last issue we faced was that the converter from binary to BCD was implemented by a very long structure. However, in logism, we implemented it after suffering a lot because there is no other option for converting. While implementing this part in Verilog, we began to search for another way to implement this converter in Verilog instead of implementing this long structure. Actually, we found a pseudocode for the double dabble converter. However, we do not use it because we did not actually understand the logic of the pseudocode and how it works. Thus, we preferred to implement the long structure for the to_bcd module in Verilog like the structure in the simulation for two reasons, which are that we basically understand the structure and how it is working, and another thing is that we were definitely sure that this structure will work correctly instead of examining other structures.

3. Important parts in the code

Implementing the SPM on Verilog, follow what is done in Logisim. For the implementation, two main modules are used to integrate all the other modules. In Logisim, these two modules are not explicitly mentioned. The two modules that are the core of Verilog implementation of the SPM are the FSM and FPGA_INT.

```
always @ (posedge clk_out or posedge rst) begin
    if (rst)begin
        Y <= 16'd0;
        out <= 16'd0; end
    else if (startOperation) begin
        out <= 16'd0;
        Y <= {{8{y[7]}}, y};//load the contents of the second input y into the Y
register for the Y register to be shifted everytime.
        end
    else if (currentState == RUNNING) begin
        out <= {product, out [15:1]};
        Y <= (Y >> 1);end
 end

 wire ywire = (currentState == RUNNING) ? Y[0] : 1'b0;
 multiplier mult (clk_out, rst, x, ywire, product);
```

This piece of code we implemented in the FSM module. This control block is central to the serial-parallel multiplication process. It orchestrates the step-by-step shifting and accumulation required to build the product. By shifting one bit of Y at a time and multiplying it with x, the design minimizes hardware usage compared to a full parallel multiplier, making it more efficient for FPGA resource-constrained environments. The use of sign extension ensures correct behavior for signed inputs, while state-based control enables clean coordination of reset, start, and run phases. Without this logic, the SPM would not be able to correctly time and sequence the multiplication process, and the results would be either incomplete or incorrect.

```
always @* begin
    if (finanode_o == 4'b1110)
        LED_out_o = Ones_r;
    else if (finanode_o == 4'b1101)
        LED_out_o = Tens_r;
    else if (finanode_o == 4'b1011)
        LED_out_o = Hundreds_r;
    else if (finanode_o == 4'b0111)
        LED_out_o = (fout_w[15]) ? 7'b1111110 : 7'b1111111;
end

always @* begin
    case (state_r)
        2'b00: begin
            if (l_w)       nextState_r = 2'b01;
            else           nextState_r = 2'b00;
            Ones_r    = led_w[0]; Tens_r = led_w[1]; Hundreds_r = led_w[2];
        end
        2'b01: begin
            if (r_w)       nextState_r = 2'b00;
            else if (l_w) nextState_r = 2'b10;
            else           nextState_r = 2'b01;
            Ones_r    = led_w[1]; Tens_r = led_w[2]; Hundreds_r = led_w[3];
        end
        2'b10: begin
            if (r_w)       nextState_r = 2'b01;
            else           nextState_r = 2'b10;
            Ones_r    = led_w[2]; Tens_r = led_w[3]; Hundreds_r = led_w[4];
        end
        default: nextState_r = 2'b00;
    endcase
end
```

For the second module, FPGA_INT, there are two always blocks considered for assigning different digits to the four 7-segment displays using a ring counter, and a Moore FSM with three different states is used for shifting digits to the left and right. Thus, this module is critical for output visualization and user interaction. Since the multiplication result can have more digits than the number of available 7-segment displays, this scrolling mechanism allows the user to view all digits of the result in segments. Without this functionality, only a limited portion of the result would be visible, making it difficult to verify or use. Furthermore, this logic demonstrates efficient use of limited hardware resources (by reusing a small number of displays) and adds an interactive component through button-controlled navigation. It also handles sign display, ensuring the correct representation of negative results. Overall, this part significantly improves the usability and completeness of the SPM system implemented on an FPGA.

Finally, the last used module in the Verilog implementation is the ALU. It was implemented as a bonus to integrate this SPM with the ALU operations to be used in a useful hardware later. Due to the limited number of switches in the FPGA, this module was thoroughly tested using a dedicated testbench, which supplied a variety of signed input values to verify the correctness of operations. The testbench evaluated several key functionalities, including the sum and difference of the inputs, their multiplication using the Serial-Parallel Multiplier (SPM), as well as bitwise AND and XOR operations. These tests ensured that the module handled signed arithmetic correctly and produced accurate results across all supported operations.

3. validation activities

1. Module-Level Validation (Unit Testing):

Test each module independently in simulation:

Multiplier Shift Register:

- Confirm right-shift per clock cycle.
- Ensure LSB is exposed correctly for each cycle.

Product Register:

- Check proper accumulation of partial products.
- Test edge cases: all zeros, all ones, max/min 8-bit values.

CSA (Carry-Save Adder):

- Test multiple sets of partial products to ensure no carry propagation.
- Verify sum and carry outputs against manual calculation.

TCMP (Two's Complement Unit):

- Validate correct conversion of negative to two's complement and back.
- Check with both even and odd values, edge cases like -128.

Sign Logic:

- Test for all sign combinations (positive × positive, positive × negative, etc.).
- Ensure the correct result and correct display sign.

to_bcd and Unit Digit Extractor:

- Verify binary to BCD conversion correctness using known values.
- Check correctness on 7-segment outputs.

2. Timing Validation

- Ensure all flip-flops are triggered on correct clock edges.
- Verify there's no race condition or glitches in data paths.
- Check FSM state transitions with proper delays.

3. Integration Testing

Once individual modules are verified, simulate the **entire pipeline**: We have different testbench scenarios to check if the all cases including edge cases are working

4. Hardware Validation (On Basys 3 FPGA)

Switch Input & Button Control:

- Validate that switches correctly reflect signed 8-bit values.
- Push button response: debounce logic or stable triggering.
- Ensure FSM properly triggers based on button states.

7-Segment Output:

- Check correct decimal representation of final result.
- Make sure negative sign shows only when expected.
- Display updates only after done state.

Contributions:

The development of the Serial-Parallel Multiplier (SPM) was a collaborative effort, and each team member contributed to different aspects of the design, coding, testing, and integration.

Nour:

- Developed the RTL structure and implemented the initial Finite State Machine (FSM) for controlling the multiplier workflow.
- Conducted the initial integration of the system with the Basys 3 FPGA board..
- Fully implemented the Carry Save Adder (CSA) module in Verilog.

Rawan:

- Designed all the Logisim modules, except for the BCD-related logic.
- Coded key Verilog modules including the Multiplier, Two's Complement (TCMP) unit, Counter, Detector and resolved multiple issues encountered during the hardware testing and debugging phase.

Doha:

- Developed Logisim modules for the BCD converter, add-3 logic, and 7-segment display.
- Handling most of the implementation of the modules before editing. Performed a small edit to the final FSM for correctness, and conducted the final integration of all components with the Basys 3 FPGA board which this is happened in the FPGA_INT module.

Bonus Contribution:

Rawan and Doha collaborated on additional improvement which is using SPM in the ALU, leading to visualize how the multiplier could function as a core component within a larger computational unit.